7 Machine learning, neural nets, support vector machines

In recent decades a number of machine learning methods for patter recognition have been launched such as neural nets and support vector machines which will be briefly discussed in this chapter. To evaluate these methods a number of large datasets have also been brought forth, compare Table 2 and
`https://en.wikipedia.org/wiki/List_of_datasets_for_machine_learning_research`
for more details.

Table 2: Datasets of images and videos for tasks such as classification, object detection and face recognition

Dataset name	Brief description	Instances	Format	Default task	Created
MNIST	Handwritten digits	60 000 + 10 000	Images, text	Classifcation	1998
CIFAR-10	Images of 10 classes of objects	60 000	Images	Classification	2009
CIFAR-100	Images of 100 classes of objects	60 000	Images	Classification	2009
KITTI	Images and videos obtained from cars	>100GB of data	Images, text	Classification, object detection	2012
SVHN	Street View House Numbers	73 257 + 26 032	Images	Classification	2011
FERET	Face Recognition Technology	11 338 from 1 199 individuals	Images	Classification, face recognition	2003

7.1 Neural nets

Let us start with considering a neural net consisting of one input layer with n_1 units corresponding to input variables $x_i, i = 1, \ldots, n_1$, an intermediate (hidden) layer with n_2 units and an output layer with K units. For unit j in the intermediate layer we compute the so-called activation value $a_j, j = 1, \ldots, n_2$, by

$$z_j = \sum_{i=1}^{n_1} w_{ji}^{(1)} x_i + b_j^{(1)}, \tag{89}$$

$$a_j = \frac{e^{z_j}}{\sum_{j'=1}^{n_2} e^{z_{j'}}}, \tag{90}$$

for weights $w_{ji}^{(1)}$ and biases $b_j^{(1)}$. With some abuse of notation we will write

$$a_j = \sigma(z_j), \quad j = 1, \ldots, n_2, \tag{91}$$

and we call σ given by (90) and (91) the *softmax* function. From the hidden layer we proceed to the output in a similar way and we obtain neural net output variables

$f_k(k), k = 1, \ldots, K$, as

$$f_k(x) = f_k(x, \theta) = \sigma\left(\sum_{j=1}^{n_2} w_{kj}^{(2)} \sigma\left(\sum_{i=1}^{n_1} w_{ji}^{(1)} x_i + b_j^{(1)}\right) + b_k^{(2)}\right), \quad k = 1, \ldots, K, \tag{92}$$

where $x = (x_1, \ldots, x_{n_1})$ is the vector of input variables, and θ is the parameter vector of all weights, $w_{ji}^{(1)}$ and $w_{kj}^{(2)}$, and biases $b_j^{(1)}$ and $b_k^{(2)}$.

We can add now add one more hidden layer which gives a neural net with two hidden layers and output

$$f_k(x) = \sigma\left(\sum_{\ell=1}^{n_3} w_{k\ell}^{(3)} \sigma\left(\sum_{j=1}^{n_2} w_{\ell j}^{(2)} \sigma\left(\sum_{i=1}^{n_1} w_{ji}^{(1)} x_i + b_j^{(1)}\right) + b_\ell^{(2)}\right) + b_k^{(3)}\right), \quad k = 1, \ldots, K, \tag{93}$$

and it should be clear how we can extend the neural net with an arbitrary number of hidden layers.

If we for instance consider a neural net for the MNIST database it is natural to consider $n_1 = 28^2 = 784$ units in the input layer, each input unit corresponding to one pixel value, and $K = 10$ corresponding to the 10 possible digits. We note that the output variables $f_k(x)$ sum to one and we can interpret $f_k(x, \theta)$ as the probability of digit k. To classify images we can first in some way estimate the parameter θ by use of a training set. Let $\hat{\theta}$ denote the estimate of θ. To classify an image x we can then put

$$\hat{k}(x) = \text{argmax}_k f_k(x, \hat{\theta}). \tag{94}$$

The crucial step here is to obtain the estimate $\hat{\theta}$. In practice the parameter vector θ may contain several thousand components and the estimation procedure is thus quite delicate. We will now discuss possible estimation methods.

7.1.1 Parameter estimation for neural nets, regularization

Suppose that we have a training set T of $|T|$ pairs (x, y) and that the neural net output $f(x, \theta)$ should approximate y. Then we introduce a suitable loss function. Let us first consider a simple case where y and $f(x, \theta)$ are real-valued. Then we may choose the loss function

$$L(\theta, T) = \frac{1}{|T|} \sum_{(x,y)\in T} (y - f(x, \theta))^2. \tag{95}$$

Let us then consider a classification setting with K classes, for instance for MNIST classification with $K = 10$. As described above we then get as output from a neural net a probability distribution $f_k(x, \theta), k = 1, \ldots, K$, for the possible class values. For a pair (x, y) where k_c is the correct class we can define $y_k, k = 1, \ldots, K$, as

$$y_k = \begin{cases} 1 & \text{if } k = k_c \\ 0 & \text{otherwise} \end{cases} \tag{96}$$

and choose the cross-entropy loss function

$$L(\theta, T) = -\frac{1}{|T|} \sum_{(x,y)\in T} \sum_k y_k \log f_k(x, \theta). \tag{97}$$

We can minimize $L(\theta, T)$ and obtain an estimate $\hat{\theta} = \hat{\theta}(T)$. The result is then that we often get a good fit to the observations in T, but if we go to a new data set the fit is typically not so good. We say then that we get an overfit. To compensate for overfitting we can introduce a regularization term $R(\theta)$, for instance

$$R(\theta) = \sum_{i=1}^{|\theta|} |\theta_i|^2, \tag{98}$$

where we sum over all components of $\theta = (\theta_1, \ldots, \theta_{|\theta|})$. Then we estimate θ by minimizing the regularized loss function

$$\mathcal{L}(\theta; T, L, \lambda, R) = L(\theta, T) + \lambda R(\theta), \tag{99}$$

where $\lambda \geq 0$ is a tuning parameter. Note that $\lambda = 0$ corresponds to no regularization which typically gives overfitting, while a very large λ corresponds to underfitting. To choose a proper value of the tuning parameter we can evaluate the regularized loss function for a separate validation set T' of pairs (x, y) or use cross-validation.

7.1.2 Convolutional neural nets

Let $w = (w_{k\ell})$ and $g = (g_{ij})$ be matrices. The convolution $w * g$ is then defined by

$$(w * g)_{ij} = \sum_k \sum_\ell w_{k\ell}\, g_{i-k,j-\ell}, \tag{100}$$

compare Section 1.2 on image filtering.

Convolutional neural nets are particularly useful for analysis of images. Such neural nets contain layers with layer transitions of the following convolution type

$$a_{ij}^{(r+1)} = \sigma\left(\sum_{k=-p}^{p} \sum_{\ell=-p}^{p} w_{k\ell}^{(r)}\, a_{i-k,j-\ell}^{(r)} \right), \tag{101}$$

where p usually is a small positive number. We note that we use here only $(2p+1)^2$ different weights and that there is the same filter operation applied in different parts of $a^{(r)}$ here regarded as an image. The filter operation could for instance consist of finding edges in an image.

A convolution layer is often followed by a pooling layer reducing the layer size. We can for instance use a maxpool operation where a layer of pixels is divided into adjacent and non-overlapping rectangles and each rectangle is replaced in the following layer by one pixel with pixel value equal to the maximal pixel value in the rectangle.

Let us conclude this short introduction to neural nets with mentioning two recent references, both with the title 'Deep Learning' which is a current term for advanced neural nets

LeCun, Y., Bengio, Y. & Hinton, G. (2015) Deep learning. *Nature* **521**, 436–444,
which gives an overview, and

Goodfellow, I., Bengio, Y. & Courville, A. (2016) *Deep Learning*, MIT Press
`{http://www.deeplearningbook.org}` ,
giving a thorough and up-to-date coverage of the field.

7.2 Support vector machines

The following description is inspired by the more complete description in Chapter 19 of Efron, B. & Hastie, T. (2016) *Computer Age Statistical Inference*, Cambridge University Press. Suppose that we have a training set T consisting of pairs (x, y), where x is an n-dimensional column vector and $y \in \{-1, +1\}$ is a two-class indicator. To begin with we will suppose that the two classes are linearly separable in the sense that there exist a real parameter β_0 and an n-dimensional parameter vector β such that with $f(x) = \beta_0 + x^T\beta$

$$yf(x) > 0 \quad \text{for all} \quad (x, y) \in T. \tag{102}$$

We can then classify a new x-vector and predict the corresponding y-value as $\text{sign}(f(x))$. A natural question is then if we can choose β_0 and β in an optimal way. The suggested solution here is to maximize the minimal distance (margin) to the separating hyperplane $f(x) = 0$ in n-space. The solution to this problem turns out to be to find

$$\max_{\beta_0,\beta} \left\{ M : \text{subject to } \frac{1}{||\beta||} y(\beta_0 + x^T\beta) \geq M \text{ for all } (x, y) \in T \right\}, \tag{103}$$

where $||\beta||$ is the Euclidean (quadratic) norm in n-space. An equivalent somewhat simpler formulation is to find

$$\min_{\beta_0,\beta} \left\{ ||\beta|| : \text{subject to } y(\beta_0 + x^T\beta) \geq 1 \text{ for all } (x, y) \in T \right\}. \tag{104}$$

In general we can not expect to find a hyperplane giving complete separation between the two classes. Then we can instead find a minimum with a regularized loss function

$$\min_{\beta_0,\beta} \left\{ \sum_{(x,y)\in T} [1 - y(\beta_0 + x^T\beta)]_+ + \lambda||\beta||^2 \right\}, \tag{105}$$

where $[a]_+$ denotes the positive part of a real number a. For linearly separable classes one can show that $\lambda = 0$ gives the previously described solution which is determined by a few points close to the separating boundary. Increasing λ corresponds to taking account of more and more data points. Similar as for neural nets one can find an optimal tuning parameter λ by use of a separate validation set or by cross-validation.

For a multiclass classification problem we can for instance for each class make a two-class classication versus the union of all other classes and then for a new observed x-vectoer to choose the class giving the largest margin. Another possibility is to consider voting for all pairwise comparisons and for a new observation to choose the class that gets that the maximal number of votes.

7.2.1 Support vector machines with kernel functions

One can show that for a new vector x to be classified one can write the classifier on the form

$$f(x) = \beta_0 + x^T\beta = \beta_0 + \sum_{i=1}^{|T|} \alpha^i x^T x^i, \tag{106}$$

where $x^1, \ldots, x^{|T|}$ are the x-vectors in the training set T and $\alpha^1, \ldots, \alpha^{|T|}$ are real parameters. This representation allows us to use a modified classifier of the form

$$f(x) = \beta_0 + x^T\beta = \beta_0 + \sum_{i=1}^{|T|} \alpha^i k(x, x^i), \tag{107}$$

where $k(u, v)$ is a positive-definite kernel function, for instance the Gaussian kernel

$$k(u, v) = \mathrm{e}^{-||u-v[[^2}. \tag{108}$$

Use of kernel functions implies possibilities of nonlinear transformations of the x-vectors and adds considerable flexibility to support vector machines.

8 Warping and matching

An important problem in analysis of multiple images is to match objects in different images. Thus we would like to know which spots in the 2D gel electrophoresis images in Figures 9 and 10 that correspond to each other, that is gauge the expression of the same protein. Similarly we want to match objects in Figures 13 and 14 in to order to be able to follow the diffusing particles and to estimate the diffusion coefficient of their motion. There is, however, a fundamental difference between these two problems. The diffusing particles move independently of each other except for the rare occasions when they come very close in all three dimensions. Thus displacements of particles that are close in the two-dimensional images are essentially independent of each other. In contrast, displacements of nearby spots in the electrophoresis images are highly correlated. The matching of objects in these two situations therefore demand quite different methods. In the present section we shall study warping methods which are useful for matching of objects in images such as the 2D gel images.

Suppose that we have a reference image $Y = Y(x)$ and another image Y' that we want to warp (transform) into Y as closely as possible according to some criterion by transforming locations such that $Y(x')$ is close to $Y(x)$. Here we regard x and x' as 2-dimensional column vectors and put

$$x' = f(x) \tag{109}$$

for some *warping function* f. For the affine warping function we have

$$x' = Ax + b = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}. \tag{110}$$

A special case of the affine transformation is the Procrustes transformation for which

$$x' = \begin{bmatrix} c\cos\theta & c\sin\theta \\ -c\sin\theta & c\cos\theta \end{bmatrix} x + b. \tag{111}$$

A special case of the Procrustes transformation consists of a dilation (scale change with a fixed factor c) and a translation

$$x' = \begin{bmatrix} c & 0 \\ 0 & c \end{bmatrix} x + b = cx + b, \tag{112}$$

and another special case of the Procrustes transformation consists of a rotation and a translation,

$$x' = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} x + b. \tag{113}$$

A simple nonlinear warping is the bilinear transformation

$$\begin{aligned} x_1' &= a_{11}x_1 + a_{12}x_2 + c_1x_1x_2 + b_1 \\ x_2' &= a_{21}x_1 + a_{22}x_2 + c_2x_1x_2 + b_2. \end{aligned} \tag{114}$$

We note that for fixed x_2 the bilinear transformation of x_1' is linear in x_1 (with slope and intercept depending on x_2) and, similarly, for fixed x_1 the transformation of x_2' is linear

in x_2. This means that an axes-parallell rectangle in the x_1x_2-plane is transformed into a polygon with four corners in the $x_1'x_2'$-plane.

Another nonlinear warping function is the perspective transformation

$$\begin{aligned} x_1' &= (a_{11}x_1 + a_{12}x_2 + b_1)/(c_{11}x_1 + c_{12}x_2 + 1) \\ x_2' &= (a_{21}x_1 + a_{22}x_2 + b_2)/(c_{21}x_1 + c_{22}x_2 + 1). \end{aligned} \tag{115}$$

The perspective transformation may be used for matching the tree tops in Figures 2 and 4. Note that both the bilinear and the perspective transformations are generalisations of the affine transformation (110).

To choose parameters of a warping transformation $x' = f(x)$ we may consider a criterion function such as

$$L(Y', Y, f) = \sum_x (Y'(x') - Y(x))^2 + \lambda D(f) \tag{116}$$

where $D(f)$ is a distortion measure of the warping function f and λ is a constant determining the balance between closeness of matching and distortion. The distortion measure could for instance measure the deviation from linearity of the warping function, and could be a sum of squared second derivatives of f.

A useful type of warping consists af a net of locally bilinear transformation. This method is used in Glasbey and Mardia (2001) to warp images fish, haddock and whiting, see Figure 40, into each other. Similarly it is used in Gustafsson et al. (2002) to match 2D gel electrophoresis images such as those in Figures 9 and 10 into each other.

For reviews of image warping methods, see Glasbey and Mardia (1998, 2001).

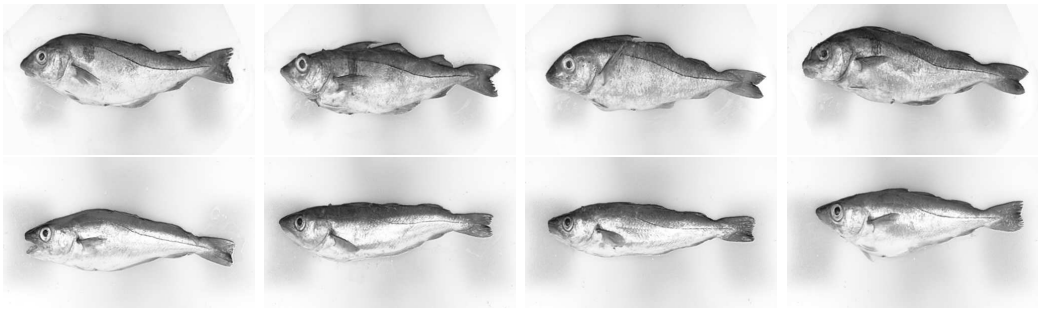

Figure 40: Images of fish warped into each other in Glasbey and Mardia (2001). Haddocks in the upper row and whitings in the lower row.